MISO: Mixed-Integer Surrogate Optimization Code Documentation

Juliane Müller
juliane.mueller2901@gmail.com

Center for Computational Science and Engineering
Lawrence Berkeley National Laboratory, Berkeley, CA, 94720, USA

Abstract

MISO is an optimization framework for solving *computationally expensive, mixed-integer, black-box, global optimization problems*. MISO uses surrogate models to approximate the computationally expensive objective function. Hence, derivative information, which is generally unavailable for black-box simulation objective functions, is not needed. MISO allows the user to choose the initial experimental design strategy, the type of surrogate model, and the sampling strategy. This code manual describes the MATLAB code and how you can use MISO to solve your optimization problems.

1 Introduction

This documentation accompanies the MATLAB implementation of the MISO (Mixed-Integer Surrogate Optimization) framework. We implemented and tested MISO in MATLAB 2012a [6]. MISO is a derivative-free surrogate model algorithm that aims at solving optimization problems of the following type:

$$\min f(\mathbf{x}) \tag{1a}$$

$$-\infty < x_i^l \leq x_i \leq x_i^u < \infty, i = 1, \ldots, d \tag{1b}$$

$$x_j \in \mathbb{Z}, \forall j \in \mathbb{I} \subset \{1, 2, \ldots, d\}, \tag{1c}$$

where x_i^l and x_i^u are the variable lower and upper bounds of variable x_i, d is the problem dimension, and $\mathbb{I}$ contains the indices of the variables that have integer constraints.

The optimization problem has the following characteristics:

- The objective function is computationally expensive to compute. Obtaining a single function value is extremely time consuming and takes several minutes to hours.
- The objective function is a black box. No analytical description of the function is available, it is, for example, a computer simulation.

- Derivatives of $f(\mathbf{x})$ are not available.
- The objective function is deterministic. The value of $f(\mathbf{x})$ is the same for the same variable input vector.
- Some variables have integer constraints: mixed-integer variables.[1]
- The objective function is multimodal. The domain scientists may have some idea of whether or not there are several basins of attraction. For black-box problems, it is not possible to tell a priori what the shape of the objective function is, and in order to avoid becoming trapped in a local minimum, we have to assume that $f(\mathbf{x})$ is multimodal and apply a global optimization algorithm.

The goal is to find the global minimum of $f(\mathbf{x})$ by doing only very few evaluations of $f(\mathbf{x})$ in order to keep the optimization time acceptable. If your objective function evaluations are computationally inexpensive, MISO will not be an efficient solver. We developed MISO for problems whose function evaluation time allows only for several hundred evaluations.

In this code companion, we focus mostly on explaining what the individual m-function do and how to use MISO for solving your optimization problems. We recommend reading the paper "MISO: mixed-integer surrogate optimization framework" by J. Müller (2015, to appear in *Optimization and Engineering*, DOI: 10.1007/s11081-015-9281-2) for further explanations and references.

MATLAB version and toolboxes: MATLAB 2012(a) and newer (tested until 2014(b)); Optimization toolbox; Global optimization toolbox; Statistics and Machine Learning toolbox.

Make sure that the MISO code directory is known to the MATLAB search path. To test the algorithm, type in the MATLAB command window

```
testdriver
```

This runs a computationally cheap test problem and should finish successfully.

We organized this code manual as follows. Section 2 is a brief overview of how surrogate model algorithms work in general and radial basis functions (RBFs). The individual m-functions of the algorithm are described in Section 3. MISO comes with several options for the initial experimental design, the type of RBF surface that can be used as surrogate for the expensive objective function, and sampling strategies (i.e., how to iteratively select new trial points for doing the expensive evaluations). These settings are defined by input parameters that are described in Section 3.1. The output of the algorithm is described in Section 4. An example of how to define your own optimization problem and how to call the algorithm is given in Section 5.

Lastly, should you encounter difficulties or bugs, please feel free to contact me at

juliane.mueller2901@gmail.com

[1] Note that MISO works also for problems with only continuous variables. Selected sampling strategies work also for problems with only integer variables.

2 Surrogate Model Algorithm and Radial Basis Functions

Surrogate model algorithms generally follow the steps described in Algorithm 1 and illustrated in Figure 1.

Algorithm 1 General Surrogate Model Algorithm

1: Create an initial experimental design and do the expensive objective function evaluations at the selected points. Fit the surrogate model.
2: Use the information from the surrogate model to select the point $\mathbf{x}_{\text{new}}$ for doing the next expensive function evaluation. Do the expensive evaluation at $\mathbf{x}_{\text{new}}$: $f_{\text{new}} = f(\mathbf{x}_{\text{new}})$.
3: Update the surrogate model and go to Step 2.
4: Stop when the stopping criterion is satisfied and return the best solution found.

First, an initial experimental design is created and the computationally expensive objective function is evaluated at the selected points. In general, any initial design strategy may be used, but it has to be ensured that there are sufficiently many points to compute the parameters of the chosen surrogate model. The objective function value predictions of the surrogate model at unsampled points are used to select the next evaluation point. After the new function value has been obtained, the surrogate model is updated if the stopping criterion has not been satisfied (for example, the budget of function evaluations has not been exhausted) and a new point is selected for evaluation. Otherwise, the algorithm stops and returns the best solution found.

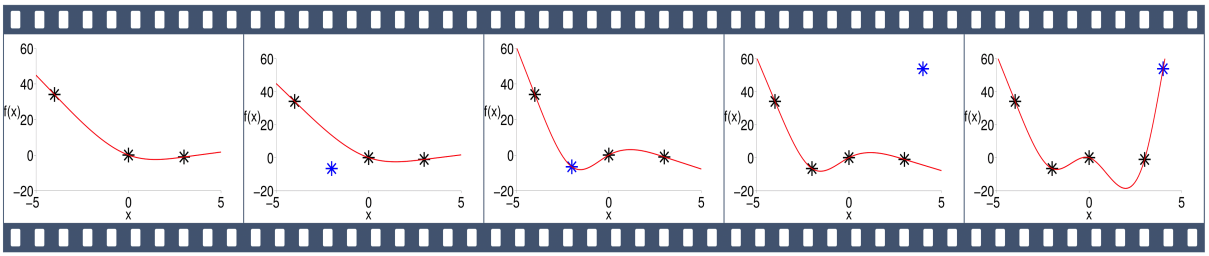


Figure 1: Illustration of the surrogate model algorithm steps described in Algorithm 1.

Although in general any type of surrogate model can be used within our MISO framework, we implemented MISO with RBF models only. Since we consider deterministic objective functions, we do not want to use non-interpolating surrogate models such as polynomial regression models or multivariate adaptive regression splines (MARS, [1]) because we would like that the surrogate model makes accurate predictions at already evaluated points. One interpolating surrogate model is kriging [4, 5]. Kriging has the advantage that it gives an uncertainty estimate together with the objective function value prediction. However, for large dimensional problems ($d > 10$), the computation time of the kriging model parameters increases significantly, rendering the model inefficient. Hence,

we implemented in MISO three RBF model types (cubic, linear, thin-plate spline). The RBF interpolant is defined as

$$s(\mathbf{x}) = \sum_{\iota=1}^{n} \lambda_\iota \phi(\|\mathbf{x} - \mathbf{x}_\iota\|) + p(\mathbf{x}), \tag{2}$$

where $\phi(\cdot)$ is the radial basis function (types defined in Table 1), $\mathbf{x}_\iota, \iota = 1, \ldots, n$, denotes the points at which the objective function value is known (already evaluated points), and $p(\cdot)$ denotes the polynomial tail whose order depends on the chosen RBF type (see Table 1). The parameters $\lambda_\iota \in \mathbb{R}, \iota = 1, \ldots, n$, and the parameters of the polynomial tail $\beta_0, \beta_1, \ldots \in \mathbb{R}$ are determined by solving the following linear system of equations

$$\begin{bmatrix} \mathbf{\Phi} & \mathbf{P} \\ \mathbf{P}^T & \mathbf{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{\lambda} \\ \boldsymbol{\beta} \end{bmatrix} = \begin{bmatrix} \mathbf{F} \\ \mathbf{0} \end{bmatrix}, \tag{3}$$

where $\Phi_{\iota\nu} = \phi(\|\mathbf{x}_\iota - \mathbf{x}_\nu\|)$, $\iota, \nu = 1, \ldots, n$, $\mathbf{0}$ is a matrix with all entries 0 of appropriate dimension, and

$$\mathbf{P} = \begin{bmatrix} \mathbf{x}_1^T & 1 \\ \mathbf{x}_2^T & 1 \\ \vdots & \vdots \\ \mathbf{x}_n^T & 1 \end{bmatrix}, \quad \boldsymbol{\lambda} = \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_n \end{bmatrix} \quad \boldsymbol{\beta} = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_d \\ \beta_0 \end{bmatrix}, \quad \mathbf{F} = \begin{bmatrix} f(\mathbf{x}_1) \\ f(\mathbf{x}_2) \\ \vdots \\ f(\mathbf{x}_n) \end{bmatrix}. \tag{4}$$

The matrix in (3) is invertible if and only if $\text{rank}(\mathbf{P}) = d + 1$ [7].

Table 1: Radial basis function types and their corresponding minimal degree μ_p of $p(\mathbf{x})$, where $\Pi^d_{\mu_p} := \{0\}$ for $\mu_p = -1$.

Name	$\phi(r) =$	μ_ρ
Linear	r	0
Cubic	r^3	1
Thin plate spline	$r^2 \log r$	1

3 Description of Individual m-functions

3.1 miso.m

The main function from which to run the algorithm is miso.m. miso.m takes seven (7) input arguments shown in Table 2, out of which only the first is mandatory. There are default values for the remaining inputs.

[x_opt, f_opt] = miso('datafile', maxeval, 'surrogate', n_start, 'init_design', 'sampling', own_design)

The first input argument (datafile), is a string with the name of the m-file in which your optimization problem is defined. We recommend using one of the examples that come

with MISO and using it as template for writing your own datafile. If you want to give only selected input arguments to miso.m, use [ ] for the arguments that you want the algorithm to assign default values to. The individual input options that can be given by the user are defined in the following subsections.

miso.m first checks which input arguments are defined and assigns default values to input arguments that were not defined by the user. miso.m then generates an initial experimental design ('init_design') (or uses the user's specified design own_design), after which the desired sampling strategy ('sampling') is called. The algorithm keeps sampling until the maximum number of function evaluations (maxeval) has been reached. The sampling strategy outputs an updated structure array Data which will then be saved as sol in the file results.mat. A progress plot that shows the development of the objective function value will be drawn. miso.m's outputs are x_opt (the best point found during the optimization procedure) and f_opt (the best objective function value encountered).

3.1.1 Input: datafile

The datafile must have as output argument a structure array Data (function call: function Data = your_filename). Within the datafile, you have to define the lower (Data.xlow) and upper (Data.xup) variable bounds for *each* variable, the problem dimension (Data.dim), the variable indices of integer (Data.integer) and continuous (Data.continuous) variables, and the function handle for the objective function (Data.objfunction). For example,

```
Data.xlow = [-10, 0, 5];
Data.xup = [0, 20, 15];
Data.dim = 3;
Data.integer = [1, 2]; (variables 1 and 2 have to be integers)
Data.continuous = 3; (variable 3 is a continuous variable).
Data.objfunction = @(x) function_handle(x).
```

The objective function must be defined such that the input variable vector x is a row vector and it must return a scalar value, i.e., each variable input vector within the variable lower and upper bounds must be evaluable and the function value has to be in $\mathbb{R} \setminus \{\infty, -\infty\}$.

3.1.2 Input: maxeval

maxeval has to be an integer number. It defines the maximum number of allowable function evaluations. In order to fit an RBF model, we need at least $d + 1$ function evaluations, and therefore maxeval has to be larger than $d+1$. If maxeval is not given by the user, the default value maxeval = 50d is used. The algorithm stops after the maximum number of function evaluations has been reached.

3.1.3 Input: surrogate

The input surrogate has to be a string that defines the type of radial basis function that is used. The options are 'rbf_c' (cubic RBF), 'rbf_l' (linear RBF), and 'rbf_t' (thin-plate spline RBF). If surrogate is not given by the user, the default value 'rbf_c' is used.

Table 2: Parameter inputs for miso.m.

Input #	Name	Description
1	datafile	string, mandatory, name of the file containing the user's problem definition
2	maxeval	integer, optional, maximum number of allowed function evaluations, default $50d$
3	surrogate	string, optional, name of the surrogate model to be used, default 'rbf_c'
4	n_start	integer, optional, number of points in initial experimental design, default $2(d+1)$
5	init_design	string, optional, name of initial experimental design, default 'slhd'
6	sampling	string, optional, strategy for iteratively selecting the sample points, default 'cptvl'
7	own_design	matrix, optional, partial or complete initial experimental design points, default [ ]

3.1.4 Input: n_start

The input argument n_start is an integer that defines the number of points to be used in the initial experimental design. If no additional initial design points are given by the user as input (init_design='slhd' or init_design='lhs'), n_start has to be at least $d+1$ (the minimum number of points needed to fit an RBF model). The default value is n_start = d+1.

3.1.5 Input: init_design

init_design defines the type of initial experimental design. The options are 'slhd' (symmetric Latin hypercube design), lhs (MATLAB's out-of-the-box Latin hypercube design lhsdesign.m), and 'own' (which indicates that you supply a set of points as initial experimental design in the input argument 'own_design'). The default value is 'slhd'. If you have a good idea of where interesting staring points might be located, you can set the init_design option to 'own' and give a matrix of initial design points in the input argument own_design (see also Section 3.1.7). The number of points you supply does not necessarily have to be $d+1$. If you supply fewer than $d+1$ points, we generate the remaining points by the symmetric Latin hypercube sampling.

3.1.6 Input: sampling

The input sampling defines the strategy according to which we iteratively select a new sample point. In MISO, we select one new sample point in each iteration. The sampling strategy options are

- 'cp': Coordinate perturbation strategy. Generate a large set of candidate points by adding random perturbations to randomly selected variable values of the best point

found so far. Generate a second set of candidate points by uniformly selecting points from the whole variable domain. Round integer variables. Score each candidate point based on its function value predicted by the RBF and its distance to the previously evaluated points. Select the point with the best score.

- 'tv': Target value strategy. Define a target value for the objective function and minimize a bumpiness measure in order to select the point in the variable domain at which it is most likely that this target value will be assumed. Use a genetic algorithm to minimize the bumpiness measure to obtain an integer-feasible solution (uses MATLAB's ga.m).

- 'ms': Minimum point of surrogate surface strategy. Use a genetic algorithm (ga.m) to find the minimum point of the surrogate surface and use this point as new sample point for the expensive evaluation. Multi-level single linkage and a genetic algorithm ensure that the minimum point will satisfy the integer constraints. Note, this method may get trapped in a local minimum of the surrogate surface which is not necessarily a stationary point of the true function.

- 'rs': Random generation of candidate points. This method is similar to cp, but instead of perturbing only a subset of the variables of the best point found so far, we *perturb all variables* of the best point found so far. We round the integer variables. No uniformly generated candidate points are used. Score each candidate point based on its function value predicted by the RBF and its distance to the previously evaluated points. Select the point with the best score.

- 'cptv': A combination of cp and tv. Whenever one sampling method fails to find improved solutions in several consecutive iterations, we switch to the other sampling method.

- 'cptvl': A combination of cp, tv, and a local search (MATLAB's fmincon.m) on the continuous variables. If neither cp nor tv are able to find improved solutions, we do a local search. We fix the integer values of the best point found so far and use fmincon.m to minimize the true function $f(\mathbf{x})$ with respect to the continuous variables only.

Which sampling method should you use? In our numerical study, we found that 'cptv' and 'cptvl' (which is the default) perform generally best. 'cptvl' tends to find higher accuracy solutions due to the local search on the true surface.

All sampling strategies can be used for problems with only continuous variables. For problems with only integer variables, the sampling methods tv, cptv, ms, and rs can be used. cp and cptvl can not be used for pure integer problems. Moreover, MISO will not work for pure integer problems in which the number of possible solutions is less than the maximum number of allowed function evaluations because the algorithm will not evaluate the same point more than once. Note, however, that we developed MISO primarily for mixed-integer problems and we did not do a numerical study in which we compare MISO with other algorithms developed for continuous and integer problems, respectively.

3.1.7 Input: own_design

If you would like to give the algorithm sample points to include in the initial experimental design, set init_design = 'own'. The input own_design has to be a matrix with d (dimension) columns and m (number of user given sample points) rows, i.e., each row is a sample point to be included in the initial design. If $m < d + 1$, the algorithm will use the 'slhd' design in order to select the remaining starting points ($d + 1 - m$ additional points in case you have not defined n_start, $\max\{d + 1 - m, \text{n\_start}\}$ additional points in case you defined n_start in the input).

3.2 slhd.m

slhd.m selects n_start points as initial experimental design by generating a symmetric Latin hypercube deign [10]. The input parameter is the structure array Data that contains the information about how many starting points are needed and what the problem dimension is. The output is a matrix with d (dimension) columns and n_start rows.

3.3 cp.m

This function is used when sampling = 'cp' (the coordinate perturbation sampling strategy). First, various parameters and counters are defined, for example, a perturbation range for generating candidate sample points and the maximum number of failed improvement trials after which the perturbation range is decreased. At the beginning of each iteration, the parameters of the selected RBF type are computed (calling function rbf_params.m).

In order to create the candidate points, a perturbation probability is defined. The perturbation probability decreases as the number of sample points increases, and thus the search becomes more local as the maximum number of allowed evaluations is approached. Candidate points are generated in two ways. One group is generated by perturbing the variable values of the best point found so far with the current perturbation range. The variables to be perturbed are selected with the computed perturbation probability, i.e., not all variables are perturbed. The second group of candidate points is generated by uniformly selecting points from the whole variable domain. We round the integer variable values such that every candidate point satisfies the integrality conditions. We want to select exactly one of the candidate points for doing the next expensive function evaluation (call function compute_scores.m).

We do the expensive objective function evaluation at the selected sample point. If the function value is better than that of the current best point, we update the best point found so far. Depending on whether or not the new evaluation point is an improvement, the counter for consecutively successful (or failed) trials is updated. If the newly selected point was an improvement, we check if the integer variable values changed from the previous best point. If it is the case, we update the surrogate model parameters using the new point, and we do a candidate search only on the continuous variables. We update the surrogate model parameters whenever we have obtained a new data point.

After the threshold for the number of consecutively successful improvement trials has been exceeded, we double the perturbation range for generating candidate points. If the number of consecutively failed trials exceeds its threshold, we half the perturbation range. cp.m updates the structure array Data and returns it to miso.m.

3.4 tv.m

tv.m is used when the target value strategy (see also [2]) is chosen as sampling strategy. We first set parameters for the sampling strategy such as a sample sequence and a weight pattern for computing target values. As in cp.m, we compute the parameters of the RBF model at the beginning of each iteration. Based on the iteration number, we select the sample stage which is either "Inf-step", "cycle-step/global search", or "cycle-step/local search". In each of the sample stages, a computationally cheap auxiliary problem is solved (see [3]). In the cycle steps, we minimize a "bumpiness measure" (see bumpiness_measure.m). If the solution of the auxiliary problem is not too close to an already evaluated point, we do the expensive simulation at this solution. If the solution is too close to a previously evaluated point, we do the expensive simulation at a random point selected from the whole variable domain. tv.m updates the structure array Data and returns it to miso.m.

3.5 ms.m

ms.m is the sampling function for using the minimum point of the surrogate surface as new sample point. At the beginning of each iteration, the parameters of the surrogate model are computed. We use the multi-level single-linkage algorithm (mlsl.m) in order to find the various minima of the surrogate surface. We discard all minima that are too close to previously evaluated points and do the expensive function evaluation at the remaining points. If there are no points remaining, we randomly select a point from the whole variable domain. We update the Data structure array in each iteration and return it to miso.m after the maximum number of allowed function evaluations has been reached.

3.6 rs.m

rs.m is the sampling strategy in which we create candidate points by adding random perturbations to *all variables* of the best point found so far. For this sampling strategy, we set an initial perturbation range and a minimal perturbation range. We have to set a parameter that determines how many iteratively failed (or successful) improvement trials are allowed before decreasing (or increasing) the perturbation range.

In each iteration, we compute the parameters of the surrogate model. We generate a set of candidate points by adding random perturbations to all variables of the best point found so far using the current perturbation range. We use the function compute_scores.m to select the best candidate point. We do the expensive objective function evaluation at the selected point and we update the best point found so far if necessary. We update the counters for iteratively failed and successful trials. If the failure counter exceeds its threshold, we half the perturbation range. Once the minimum perturbation range has been reached, we assume that we are in a local minimum and start the search from

scratch, i.e., we create a new initial experimental design and use the random perturbation search anew. We iterate until the maximum number of allowed function evaluations has been reached.

3.7 cptv.m

cptv.m is a hybrid of the sampling strategies cp.m and tv.m. Similarly as for cp.m and tv.m, we have to first define parameter values for the threshold of failed and successful improvement trials, etc. In each iteration, we first compute the parameters of the surrogate surface. We start the iterative sampling with the cp-method (for details, see Section 3.3), i.e., we use the cp-sampling until we have reduced the perturbation range to its predefined minimal value. Once this value has been reached, we go to the tv sampling method (for details, see Section 3.4). We stay in the tv sampling stage until the counter for failed improvement trials has reached its threshold, and we then return to the cp sampling method. We cycle between cp and tv sampling until the maximum number of allowed function evaluations has been reached.

3.8 cptvl.m

cptvl.m is a hybrid of the sampling strategies cp, tv, and MATLAB's built-in gradient-based minimization function fmincon.m. cptvl.m starts as cptv.m with the cp search. Once cp does not yield any improved solutions, we switch the search to tv. After the tv search, we go back to the cp search. At the end of the second cp search, if neither the previous tv search nor the second cp search was successful, we go to the local search. Otherwise we cycle between cp and tv until cp and then tv are consecutively unsuccessful. For the local search, we use MATLAB's function fmincon.m *on the continuous variables only*. We fix the integer variables of the best point found so far and use the continuous variables as starting guess for fmincon. fmincon searches on the true objective function and it is likely that this search will consume the remaining available function evaluations. The goal of this final local search is to find at least a local minimum that is associated with the fixed integer variables and to obtain a higher accuracy solution. cptvl.m returns the updated Data structure array to miso.m. Depending on the maximum number of allowed function evaluations and the sample stages, it is possible that the local search will not be entered.

3.9 rbf_params.m

rbf_params.m is used to compute the parameters of the chosen RBF model. This function is called by all sampling strategies. As input, we need the Data structure array (in particular the sample points and their function values) and the rbf_flag that tells us which type of RBF model is desired (cubic, linear, or thin-plate spline). The function returns the RBF parameters λ_i, $i = 1, \ldots, n$ and $\beta_0, \beta_1, \ldots, \beta_d$.

3.10 rbf_matrices.m

We need the function rbf_matrices.m whenever the target value strategy is used for selecting sample points, i.e., in tv.m, cptv.m, and cptvl.m. When using the target value strategy, we define a target value for the objective function and we solve an optimization problem

in order to find the point in the variable domain where it is "most likely" that the target value will be assumed. We find this point by minimizing a bumpiness measure for which we need the RBF matrices $\mathbf{\Phi}$ and $\mathbf{P}$ in equation (3). In rbf_matrices.m, we first compute the pairwise distances of all already sampled points to each other, we compute the RBF value of the distances based on the type of RBF model used, and we use these value to set up the matrices $\mathbf{\Phi}$ and $\mathbf{P}$.

3.11 rbfvalue.m

rbfvalue.m is called whenever the target value sampling strategy is used (tv.m, cptv.m, and cptvl.m). It computes the radial basis function value (*not* the predicted objective function value) with the chosen RBF type. The input is the distance $\|\mathbf{x}_\nu - \mathbf{x}_\iota\|_2$ between the points $\mathbf{x}_\nu$ and $\mathbf{x}_\iota$, and a flag (rbf_flag) for the RBF type. The output is the RBF value $\phi(\|\mathbf{x}_\nu - \mathbf{x}_\iota\|_2)$.

3.12 rbf_prediction.m

rbf_prediction.m is used only in connection with the target value sampling strategy (tv.m, cptv.m, cptvl.m). It computes the predicted objective function value for a given input vector. The function's input arguments are the point(s) at which the objective function value should be predicted, the Data structure array, the RBF parameters, and the indicator for the type of RBF model to use. The output is the objective function value prediction.

3.13 compute_scores.m

compute_scores.m is used by the sampling strategies that use random perturbations to create candidate points (cp.m, cptv.m, cptvl.m, rs.m). The input parameters are the Data structure array, the matrix of candidate points, the weight for the surrogate score, the RBF parameters, and the RBF type.

We discard all candidate points that are too close (closer than a preset threshold) to already evaluated points. We use the surrogate model to predict the objective function values of all remaining candidate points. This is computationally cheap. We scale the values to [0,1], where the lowest predicted function value gets 0 and the largest prediction gets 1 (see also [8]) (surrogate score). We compute the distance of each candidate point to the set of already evaluated points and scale the distance values to [0,1], where the largest distance obtains 0 and the smallest distance obtains 1 (distance score). We compute a weighted sum of the scores and select the candidate point with the best (lowest) score as new sample point. The scoring weights are repeatedly cycling through a pattern defined at the beginning of the sampling strategy. A large weight for the surrogate score puts more emphasis on the predicted objective function value. The predicted objective function values tend to be lower in the vicinity of already evaluated points that have a low objective function value. Thus, the search is more local. If the weight for the distance score is large, points that are far away from already sampled points are preferred, and thus the search is more global.

3.14 inf_step.m

The inf_step is used only in connection with the target value sampling strategy (tv.m, cptv.m, cptvl.m). We set the target value for the objective function to $-\infty$ and solve the global optimization problem of minimizing the bumpiness measure (see also [3]).

3.15 bumpiness_measure.m

bumpiness_measure.m is needed only when the target value sampling strategy is used (tv.m, cptv.m, cptvl.m). The input variables are the vector at which the bumpiness of the surface should be evaluated, the structure array Data, the target value for which the bumpiness is to be computed, and the RBF parameters and matrices. The output is the bumpiness value. For details of the bumpiness computation, see [2].

3.16 mlsl.m

mlsl.m is used only when the sampling strategy is the minimum point of the surrogate surface. We use the multi-level single-linkage algorithm [9] in order to find the points in the variable domain where the surrogate model may have local and global optima. The input parameters are the structure array Data and the RBF parameters. The output is a matrix with new sample points at which the expensive objective function will be evaluated and the RBF model's prediction of these function values.

3.17 newp.m

newp.m is called by mlsl.m and checks whether or not a newly suggested point is a new local minimum of the response surface.

4 MISO Output

miso.m returns the best point found during the optimization run and the corresponding function value. It also generates a file results.m that contains the complete sample history and the settings used by MISO. In order to access the data, type

```
load results.mat
```

into the command prompt in MATLAB (make sure the results.mat file is located in a directory known to the MATLAB search path). A structure array sol will appear in the workspace. The fields of sol are described in Table 3.

5 Example

In this section, we show an example of how to define an optimization problem and how to call miso.m to solve it. You must provide a data file (see Section 3.1.1 for the details). The data file contains all information about the optimization problem. The MISO codes come with some test functions, for example, datainput_hartman3.m. We recommend using this file as a template for defining your own problem. Section 3.1.1 shows the mandatory

Table 3: Fileds of the structure array sol.

Field name	Description
xlow	Vector with variable lower bounds
xup	Vector with variable upper bounds
integer	Vector with indices of integer variables
continuous	Vector with indices of continuous variables
objfunction	Name of objective function handle
maxeval	Maximum number of allowed function evaluations
surrogate	Name of the used surrogate model
init_design	Name of the initial experimental design strategy
sampling	Name of the sampling strategy
number_startpoints	Number of points in the initial experimental design
tol	Minimum distance between sample points
m	Number of function evaluations done
S	Matrix (m $\times$ dimension) with evaluated points
Y	m-vector with objective function values
T	m-vector with evaluation times of each point
fbest	Best function value found during optimization
xbest	Best point found during optimization
total_T	Total time needed by optimization algorithm
own_design	User given (partial) initial experimental design

problem specifications that must be given for all problems.

When defining the objective function, you have to include the command

```
global sampledata
```

This global variable collects the sample points, function values, and function evaluation times. Define your problem such that the output of your objective function definition is a scalar value y. To time the function evaluation, use the command

```
fevalt = tic;
```

before the expensive simulation is started. After the simulation is finished and your value y has been computed, use the command

```
t = toc(fevalt);
```

Last, collect the new data in the global variable sampledata by using the command

```
sampledata = [sampledata; x(:)', y, t];
```

Hence, your data input file should look similar to the code shown in Figure 2. The red boxes indicate the code lines that have to be included in the datafile.

```
function Data=datainput_hartman3

% example optimization problem (computationally cheap)
% 3-dimensional Hartman function
%-------------------------------------------------------------------------
%Author information
%Juliane Mueller
%juliane.mueller2901@gmail.com
%-------------------------------------------------------------------------
%
%Input: none
%
%Output:
%Data - structure array with all problem information
%-------------------------------------------------------------------------

Data.xlow =zeros(1,3);  % variable lower bounds
Data.xup=ones(1,3);     % variable upper bounds
Data.objfunction=@(x)hartman3(x); %handle to objective function
Data.dim = 3;   %problem dimesnion
Data.integer = [1];   %indices of integer variables
Data.continuous = [2,3]; %indices of continuous variables

end %function

function y=hartman3(x) %objective function
global sampledata; %global variable that collects sample points, function values and evaluation times
c=[1,1.2,3,3.2]'; %c,A,b are data vectors
A=[3, 10, 30; 0.1,10,35; 3, 10, 30;0.1,10,35];
P=[0.3689    0.1170    0.2673
    0.4699    0.4387    0.7470
    0.1091    0.8732    0.5547
    0.0382    0.5743    0.8828];
x=x(:)'; % make sure vector is row vector
fevalt = tic; %start timer for function evaluation
y=-sum(c.*exp(-sum(A.*(repmat(x,4,1)-P).^2,2))); %compute objective function value
t = toc(fevalt); %stop timer for function evaluation
sampledata = [sampledata; x(:)',y, t]; %collect sample data (point x, value y, time t)
end %hartman3
```

Figure 2: Example input datafile for 3-dimensional Hartman function.

After you have defined your optimization problem, you need to decide about the settings of the algorithm (input options in Section 3.1).

For using default options, simply call MISO by typing into the command prompt, for example,

```
[x_opt, f_opt] = miso('datainput_hartman3')
```

Make sure that the directory of the MISO files is known to MATLAB's search path and that your data file is in the same directory.

If you want to specify a maximum number of allowed function evaluations, say 200, call MISO as follows:

```
[x_opt, f_opt] = miso('datainput_hartman3', 200)
```

If you want to use your own initial experimental design without generating any additional points, you have to define a matrix own_design of appropriate dimensions.

```
own_design = [ 0, 0.2, 0.9; 0, 0.1, 0.45; 1, 1, 1; 1, 0.75, 0.99];
```

If you do not provide (dimension + 1) points in the own_design matrix, the missing points will be generated by the slhd method. We let MISO know about our own design by setting the initial experimental design strategy to 'own' and supplying the matrix own_design as input argument (note that the input arguments [ ] are used to tell the algorithm to use default values for the corresponding inputs, here 'surrogate', n_start, and 'sampling'):

```
[x_opt, f_opt] = miso('datainput_hartman3', 200, [ ], [ ], 'own', [ ], own_design)
```

After the algorithm has finished, you will find the file results.mat in the current MATLAB directory (see Section 4 for details). You will also see a progress plot of the development of the objective function value (similar to Figure 3).

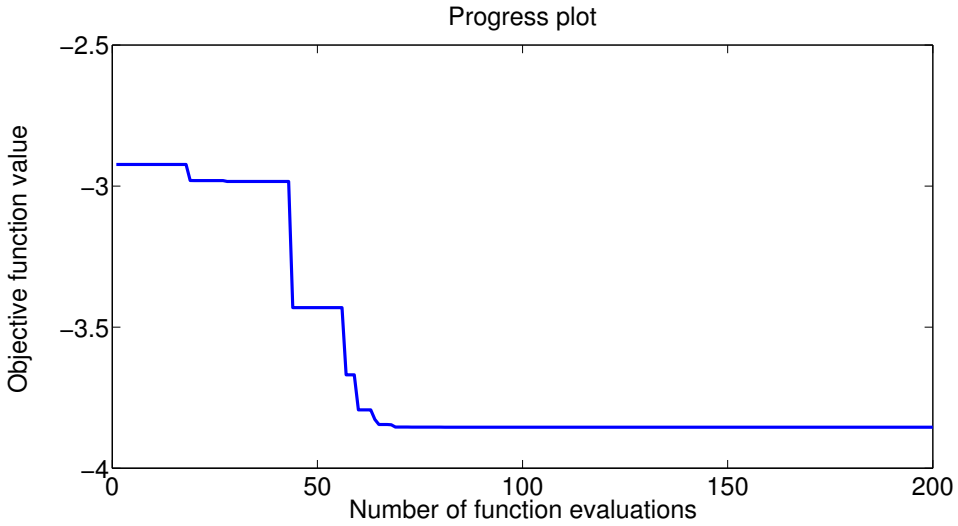
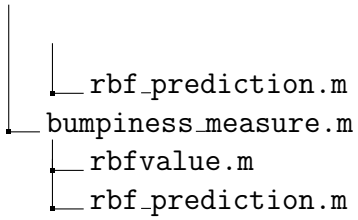


Figure 3: Progress plot for the test example datainput_hartman3.m

6 Code Structure

We outline the code structure here. Functions in subtrees indicate that they are called by a higher level function. The letters (a) - (f) indicate the sampling options.

```
miso.m
├── slhd.m / lhsdesign.m
├── (a) cp.m
│   ├── rbf_params.m
│   └── compute_scores.m
├── (b) tv.m
│   ├── rbf_params.m
│   ├── rbf_matrices.m
│   │   └── rbfvalue.m
│   ├── inf_step.m
│   │   └── rbfvalue.m
│   ├── rbf_prediction.m
│   └── bumpiness_measure.m
│       ├── rbfvalue.m
│       └── rbf_prediction.m
├── (c) ms.m
│   ├── rbf_params.m
│   └── mlsl.m
│       ├── rbf_prediction.m
│       └── newp.m
├── (d) rs.m
│   ├── slhd.m / lhsdesign.m
│   ├── rbf_params.m
│   └── compute_scores.m
├── (e) cptv.m
│   ├── rbf_params.m
│   ├── compute_scores.m
│   ├── rbf_matrices.m
│   │   └── rbfvalue.m
│   ├── inf_step.m
│   │   └── rbfvalue.m
│   ├── rbf_prediction.m
│   └── bumpiness_measure.m
│       ├── rbfvalue.m
│       └── rbf_prediction.m
└── (f) cptvl.m
    ├── rbf_params.m
    ├── compute_scores.m
    ├── rbf_matrices.m
    │   └── rbfvalue.m
    ├── inf_step.m
    │   └── rbfvalue.m
```



References

[1] J.H. Friedman. Multivariate adaptive regression splines. *The Annals of Statistics*, 19:1–141, 1991.

[2] H.-M. Gutmann. A radial basis function method for global optimization. *Journal of Global Optimization*, 19:201–227, 2001.

[3] K. Holmström. An adaptive radial basis algorithm (ARBF) for expensive black-box global optimization. *Journal of Global Optimization*, 41:447–464, 2008.

[4] D.R. Jones, M. Schonlau, and W.J. Welch. Efficient global optimization of expensive black-box functions. *Journal of Global Optimization*, 13:455–492, 1998.

[5] G. Matheron. Principles of geostatistics. *Economic Geology*, 58:1246–1266, 1963.

[6] MATLAB. *MATLAB R2012a*. The MathWorks Inc., Natick, Massachusetts, 2012.

[7] M.J.D. Powell. *The Theory of Radial Basis Function Approximation in 1990*. Advances in Numerical Analysis, vol. 2: wavelets, subdivision algorithms and radial basis functions. Oxford University Press, Oxford, pp. 105-210, 1992.

[8] R.G. Regis and C.A. Shoemaker. Combining radial basis function surrogates and dynamic coordinate search in high-dimensional expensive black-box optimization. *Engineering Optimization*, 45:529–555, 2013.

[9] A.H.G. Rinnooy Kan and G.T. Timmer. Stochastic global optimization methods, part II: multi level methods. *Mathematical Programming*, 39:57–78, 1987.

[10] K.Q. Ye, W. Li, and A. Sudjianto. Algorithmic construction of optimal symmetric Latin hypercube designs. *Journal of Statistical Planning and Inference*, 90:145–159, 2000.